ソフトゼミＢ　第２回

ＳＴＧ制作その１

ゼミＢ、ＳＴＧ（シューティングゲーム）担当市川です。ゼミＡを通して一通りプログラムが書けるようになってきたと思います。とは言え、流石に皆さんがゼロからゲームを作るのは難しいです。そこでサンプルコードを配布します。これを元に作っていきましょう。

✤ゲーム制作の概要

家や船を作るときにも設計図を作ってから作りますよね？ゲーム制作でも同じです。まず簡単にどんな仕様を実装するかまとめてみました。

まず、基本として

・自機がある
・敵機が出てくる
・敵機はランダム生成
・敵機は真下に直進
・敵機の弾も真下に直進
・敵機を倒すとアイテムが出現
・HP 制
・アイテムは取ると HP が回復かつ加点
・自機は十字キーで移動する
・自機の弾は Z ボタンで発射、真上に直進

プログラムでは文字通り書いたものしか反映されないので、こういう風にまとめておくと作りやすいかもしれません。

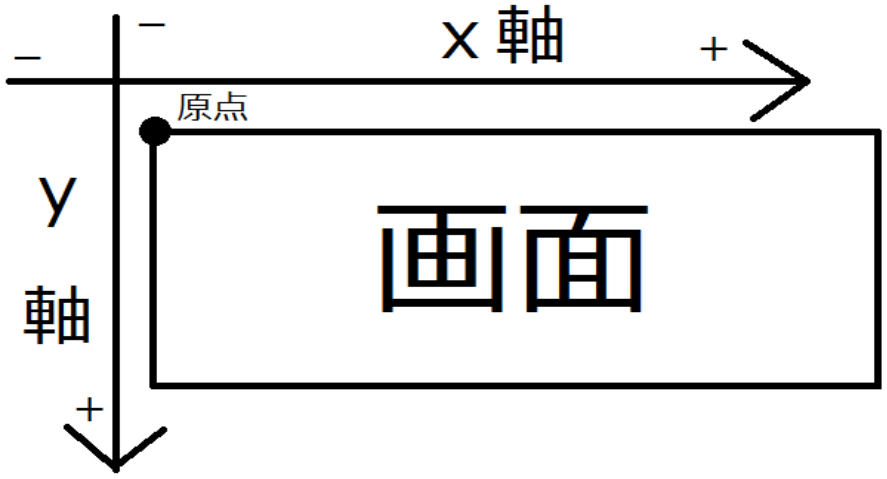
それと、座標を用いて画像の制御をするのですが、画面上の座標と画像の座標は一番左上を原点に、右方向と下方向が正になってます。数学で慣れ親しんだ形と異なるので注意してください。

Q. なんで左上が原点なの？数学と同じようにやればいいじゃん。

A. パソコンの開発がアメリカ起源なのが理由と思われます。

アメリカでは英語が使われ、プログラムも英語がベースです。英語を書くときは左から右、上から下に書きますよね？つまり、プログラムも同じように左上から右下に書いたほう

が書きやすいです。同様に、画面の出力も左上から行われる方式があります。このため左上を原点とした今の画面座標がある、と思われます。



✤ファイル分割

さて、どんなゲームを作るかは決まりましたが、実際の作業にはまだ入りません。今後ゲーム開発をしていくとプログラムが一万行を超えることはざらです。そんな中プログラム開発していると何がどこにあるか分からなくなるかもしれません。なので、今回はファイル分割という作業を行っていきます。まだゲーム制作には入れませんが、これが終わればすぐです。頑張りましょう。

まず、Visual Studio を開いたうえで、このゲームのサンプルを書いてくファイル、プロジェクトを開きます。（新しく作る場合は先日行ったＤＸライブラリの設定を忘れないこと）

プログラムを入力できる画面を開いたら、右側のソリューションエクスプローラーの「ソースファイル」又は「ヘッダーファイル」とあるところを右クリックし、「追加」から「新しい項目」を選んでください。もしくは Ctrl + Shift + A の同時押しをしてください。それぞれｃｐｐファイル、ヘッダーファイルを入れておくと良いでしょう。もし、ソリューションエクスプローラーが表示されてなかったとしても上側の「表示」から表示できます。

新しい項目を選んだら、C++ファイルとヘッダーファイルのどちらかを選ぶ画面になったと思います。今回作る C++ファイルとヘッダーファイルをまとめておいたので、同じ名前のファイルを作っていってください。

ソースファイル

・create.cpp　・draw.cpp　・initialize.cpp　・judge.cpp　・load.cpp

・move.cpp　・update.cpp　・main.cpp　・stage1.cpp

ヘッダーファイル

・define.h　・extern.h　・function.h　・global.h　・struct.h

作るファイルは以上です。新しい項目の名前の欄に、main.cpp なら「名前(N)：　main.cpp」となるように入力し、追加しましょう。

追加したら、配布資料の「stgsample2017.txt」というファイルを開いてください。そのファイルに、それぞれのｃｐｐ、ヘッダーファイルの内容が記述されているので、コピーして貼り付けてください。#pragma once　という記述は削除して大丈夫です。

配布資料の img ファイルについては、今回のプロジェクトの保存先にある.cpp などが入っているフォルダ（.sln で終わるファイルのある場所にフォルダがあります）に入れておけば大丈夫です。

✤ゲーム作成

ファイル分割のコピペも終わったら、ついにゲーム制作です。あらかじめ【　】で囲まれた穴を作ってありますので、全部で13個の穴２から埋めていきましょう。穴２はstruct.h、initialize.cpp、create.cpp、move.cpp、stage1.cpp にあります。もし穴２が見つからない時は、Ｃｔｒｌ＋Ｆキーで入力欄を出し、その下部にある「現在のドキュメント」を「ソリューション全体」にした上で「穴 2」を検索すれば出てきます。ヒントが書いてあるので、それを参考にしながら埋めていってください。また、可能な限り数字は入れず変数や define で定義された値を入れてください。

例

```
//敵の移動に関する関数
void move_enemy(){
    //すべての敵(enemy[0]～enemy[ENEMY-1(=49)])の中で
    for (int i = 0; i < ENEMY; i++){
        //存在しているものに対して
    if (enemy[i].hp != 0){
            //x,y 方向の速度を現在の座標に加える
```

enemy[i].x += 【穴 2 - 10】;　　～～以下略～～

穴埋めのヒント

穴 2-1,2-2　　struct.h にあります。座標や速度がなかったら何もできませんね。

穴 2-3,2-4　　initialize.cpp にあります。数字は入れずに、define の……

穴 2-5～7　　move.cpp にあります。x 軸、y 軸の正方向はどの方向でしょうか。

穴 2-8,2-9　　move.cpp にあります。ここは少し特殊です。画像の座標は、画像の一番左上にあります。そのうえで画像を画面外に出さないためには……

穴 2-10,2-11　move.cpp にあります。速度を表す変数は何でしょうか？

穴 2-12　　create.cpp にあります。Hp がゼロになっていることが条件です。

穴 2-13　　stage1.cpp にあります。60 フレーム毎に、というのは % を使ってみましょう。

穴 2 が埋め終わったら画面上部の「ローカル windows デバッガー」を押すか、F5 を押してください。正常に実行できれば今日は終わりです。お疲れさまでした。あたり判定なし、弾も出ない本当に動くだけのゲームですが、回を進めながら段々と実装していきましょう。

以下は今回配ったコードについての説明なので、必ず読んでおいてください。

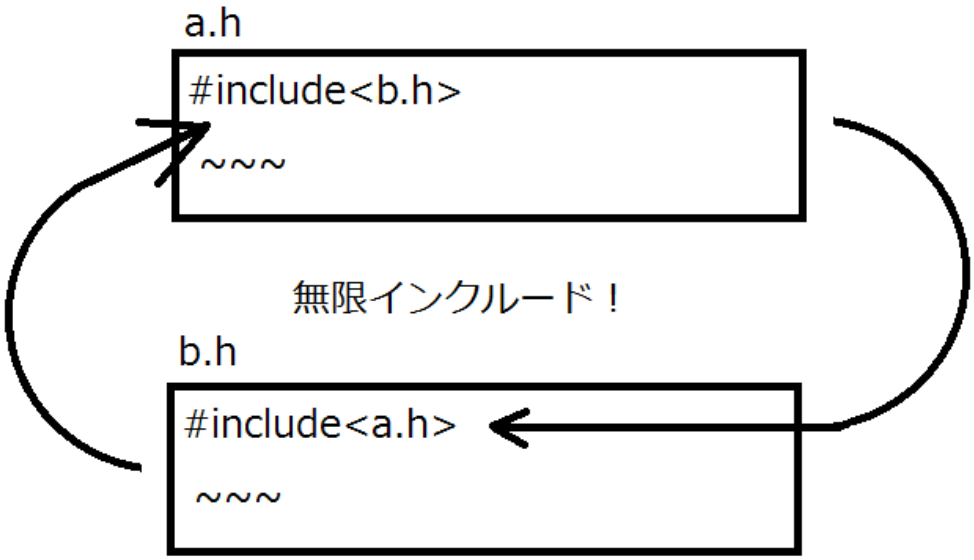
✤ヘッダーファイルを「インクルードする」とは？

コピペでＣＰＰファイルやヘッダーファイルの内容を入れていきましたが、この先頭にはたいてい#include “ ～～.h ”と書いてあります。この “ ”　又は < > で囲まれた.h で終わるファイルはヘッダーファイルと言います。ヘッダーファイルは#include でインクルードすれば、ヘッダーファイルの内容をその#include の行で読み取ってくれます。つまり、#include<stdio.h>とすればその行は stdio.h と丸ごと入れ替わる、という事です。例えば関数の宣言は新しいファイルを用意すればその都度行わなければなりませんが、あらかじめ関数の宣言のヘッダーファイルを用意すればそれをインクルードするだけで関数の宣言ができます。すごく便利です。

今回のプログラムでは math.h をインクルードしていますが、この math.h には様々な数学関数が入っています。sin や cos はもちろん、「#define _USE_MATH_DEFINES」という記述を冒頭に入れてやれば、π を利用できます。使う際はπを使いたいところに M_PI と記述しましょう。ちなみに、M_PI_4 という定義などもあり、これはπ/4 に等しいです。

ちなみにインクルードするとき、インクルードされた先のファイルでほかのファイルのインクルードがあると、そのファイルもインクルードされます。なので、あるファイルがイ

ンクルードされた先で元のファイルがインクルードされると無限ループになってエラーがおきてしまいます。



これを避けるために#pragma once という記述で、一度だけインクルードするよう設定できますが、今回は使ってません。

✤それぞれのファイルの意味

計１４のファイルがありますが、当然それぞれに意味があります。Stage1.cpp のみがゲームのステージの進行を記述し、他はすべてシステム制御用のファイルです。ほとんど名前の通りで、例えば create なら敵や弾の生成を制御する関数が入ってます。しかし、extern といった見慣れないものもあるかと思いますので、一つ一つ説明していきます。

ソースファイル

・create.cpp

敵や弾の生成を制御する関数が入ってます。

・draw.cpp

描画を制御する関数が入ってます。画像の座標は、画像の一番左上の点の座標になっています。穴 2-8,9 のあたりはそれが理由です。

・initialize.cpp

初期化用の関数が入ってます。

・judge.cpp

あたり判定を行う関数が入ってます。

・load.cpp

画像読み込み用の関数が入ってます。LoadGraph 関数については後述する補足にて。

・move.cpp

敵や自機、弾の移動を制御する関数が入ってます。

・update.cpp

敵の座標やあたり判定を実行するための関数が入ったファイルです。このファイル内の update という関数は一フレーム毎に create、judge、move 等の関数を実行し、データを更新し続けています。

・main.cpp

main 関数が入ってます。この関数はプログラムの起動から終了までが記載された、いわば全体図です。

・stage1.cpp

ステージ用のファイルです。敵の生成はこのファイルの stage1 関数で create を使って生成しています。今はまだ stage1 しかありませんが、stage2、3 と増やしていけると良いですね。

ヘッダーファイル

・function.h

関数の宣言用のヘッダーファイルです。関数の宣言が入ってます。

・global.h

・struct.h

・extern.h

この三つは三つで一組です。今回は構造体を使っています。Struct.h には構造体の定義が、global.h と extern.h には宣言が入っています。main.cpp では global.h をインクルードし、それ以外では extern.h をインクルードしてます。ようは、extern というのは複数ファイルで構造体を使うためのものです。というのも、extern を使わずに複数のファイルで構造体を使おうと struct.h の形で宣言すると、構造体が再定義され同じ変数が重複して定義されてしまいエラーになります。これを避けるために、定義を行わない宣言 extern を用いて main.cpp 以外のファイルではインクルードをしています。

Enom や GetColor については後述する補足の欄を見ておいてください。

・define.h

様々な値の設定を#define で行います。#define は、複数のファイルでも共有して使うことができます。

✤ 補足

画像を入れるための構造体 struct S_image はこうなってます。

```
//画像の struct
struct S_image{
        int img_player;     //自機の画像を格納
        int img_enemy;      //敵の画像
        int img_bullet;     //弾の画像
        int img_item;       //アイテム
        int img_gameover;   //ゲームオーバー画面
        int img_title;      //タイトル画面
        int img_background; //背景
};
```

このように、一つの画像につき一つの変数が必要です。新たに画像を足したときはここに変数を足すのも忘れないでくださいね。

列挙型 enom について

```
 enum e_flag{    //列挙型 enum, ここでは
        title,      //title = 0
        gameplay,   //gameplay = 1
        gameover,   //gameover = 2
};              //の定数を定義していると考え, これ以外の値を取らない
```

割と書いてある通りです。If(flag==title)のように、変数にしたいものを　,　で分けて並べていくだけでゼロから順番に変数を定義してくれます。Int 型で良いとか思うかもしれませんが、ソースコードが複雑になるゲーム制作では見た目の分かりやすさは大事なのです。可読性も意識していきましょう。ちなみに、gameplay=5 という風に宣言時に値を定義できますが、このときそのあとに列挙された変数は順に 6,7……と続いていきます。

DX ライブラリの関数の説明

・GetColor 関数

struct.h に

int black = GetColor(0,0,0);

という記述があります。これは Dx ライブラリの GetColor 関数を利用してます。引数は左から赤、緑、青の色の強さを示しています。

・LoadGraph 関数

load.cpp の LoadGraph という関数では画像を読み込めます。あらかじめ用意した画像の場所を指定する必要があり、このとき拡張子(.png とか)も必要なので注意です。

・ScreenFrip 関数

draw 関数の最後の方に ScreenFrip 関数があります。この関数は、裏画面描画したものを表画面に描画しなおす関数です。と言っても、よくわからないと思います。この関数は画面のちらつきを抑えるためのものです。最初から普通に描画をすると、画像の消去と描画が常に見えていることになり、これが画面のちらつきに繋がります。なので、一度実際には見えない裏画面に全部描画を終えてから、最後にまとめて描画をします。これにより、画像の消去と描画が 1 フレームに 1 度しか見えないため、画面のちらつきを防ぐことが出来ます。

・DrawRotaGraph 関数

まずは関数の宣言を見てみましょう。

int DrawRotaGraph(int x, int y, double ExtRate, double Angle, int GrHandle , int TransFlag , int TurnFlag) ;

引数は始めから、x 座標、y 座標、拡大率、回転角度、グラフィックハンドル、透明度のフラグ、左右反転フラグです。 拡大率は 1.0 で等倍、2.0 なら 2 倍、0.5 なら 1/2 倍ですね。回転角度は弧度法です。πなら丁度反対向き、π/2 なら時計回りに 90 度です。そして、DrawGraph 関 数では座標は左上だったと思いますが、こちらは画像の中心を指定されます。

DrawRectGraph 関数もあり、こちらは画像ファイルを指定した範囲を表示できます。

さらに、二つ合わせた DrawRectRotaGraph 関数というのもあります。こちらはＤＸライブラリの HP には載っていないので注意が必要です。調べれば出てきます。

このほかにもゲームを作るにあたって様々な便利な関数がＤＸライブラリに入っています。DX ライブラリのＨＰで何があるかを確認できるのでぜひ見ておいてください。